Problem 1. [5 points] Arrays and Linked Lists: Read the assigned chapters and notes for Module

3 located in the Learning Activities area. Then provide solutions to the following:

(a) [3 points] Download the List.zip file and then modify List.cpp to implement details for

the predicate function called isInList() . The prototype for this function is as follows:

**File included in Zip**

(b) [2 points] Write a brief summary explaining the differences between an array-based list

and a linked list data structure. Describe the advantages and disadvantages of both types

of lists in comparison to each other.

**A linked-list structure takes a base structure element and links additional structure elements together as the program requires, while an array-based list allocates its entire list in its declaration. A linked-list can grow as large as memory permits, useful when the user is unsure of size of input. However an array-based list can only grow as large as it is defined. A linked-list can also store elements of different sizes, while an array-based list allocates equal-space for each element in the array.**

Problem 2. [5 points] Stacks and Queues: Read the assigned chapters and notes for Module 4

located in the Learning Activities area, then provide solutions to the following:

(a) [3 points] Download the Stack.cpp file. Modify the to implement details for the helper

function called reverseStack() . The prototype for this function is as follows:

**File included in Zip**

(b) [2 points] When considering the push() and pop() operations for a stack, briefly describe

how a compiler program could use a stack to implement delimiter matching. For example,

matching delimiter strings could be: “{”, “}”, “(”, “)”, and “/\*”, and “\*/”.

**A compiler can use a stack’s push operation on “{“, “(“, and “/\*”, and any following ‘open’ delimiter strings will be added ‘on top’ of the stack with additional ‘pushes’. When the compiler reaches a ‘closing’ delimiter string, the compiler will ‘pop’ its matching ‘open’ delimiter string from the stack.**